PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46

A1

(11) International Publication Number: **WO 99/44132**

(43) International Publication Date: 2 September 1999 (02.09.99)

(21) International Application Number: PCT/US99/03944

(22) International Filing Date: 24 February 1999 (24.02.99)

(30) Priority Data:

60/076,048	26 February 1998 (26.02.98)	US
09/044,790	20 March 1998 (20.03.98)	US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, MS UPAL01-521, Palo Alto, CA 94303 (US).

(72) Inventors: WOLLRATH, Ann, M.; 9 Northwoods Road, Groton, MA 01450 (US). JONES, Peter, C.; 85 Bacon Street, Winchester, MA 018900 (US).

(74) Agents: GARRETT, Arthur, S.; Finnegan, Henderson, Farrabow, Garrett & Dunner, L.L.P., 1300 I Street, N.W., Washington, DC 20005-3315 (US) et al.

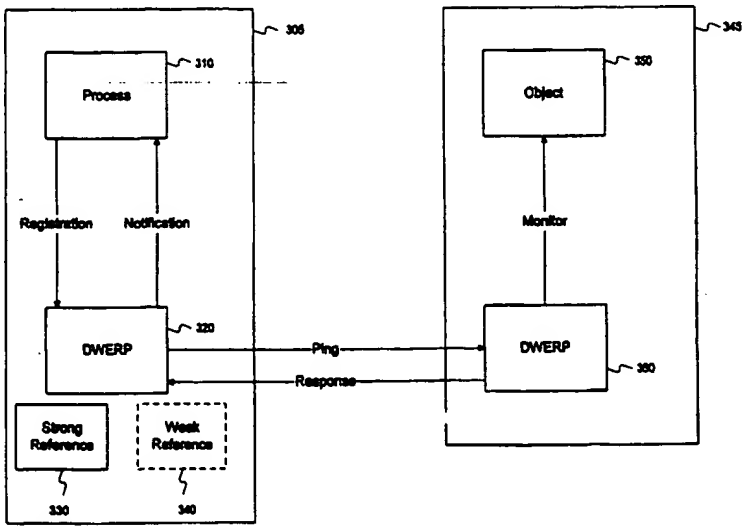
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE). OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR DETERMINING STATUS OF REMOTE OBJECTS IN A DISTRIBUTED SYSTEM



(57) Abstract

Systems consistent with the present invention, a method and apparatus are provided for selectively supplying a state change associated with remote objects in a distributed system. The method involves registering a request from a computational entity to receive notification as to a state change associated with a remote object. Registration of a notification request causes the creation of a remote weak reference to the remote object including an identifier of a location of the remote object. Periodically, a request is sent to a location based on the identifier of the remote weak reference. When it is determined that a state change associated with the remote object has occurred, the registered computational entity is notified accordingly.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR DETERMINING STATUS OF REMOTE OBJECTS IN A DISTRIBUTED SYSTEM

RELATED APPLICATIONS

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application.

Provisional U.S. Patent Application No. 60/076,048, entitled "Distributed Computing System," filed on February 26, 1998.

U.S. Patent Application No. 09/044,923, entitled "Method and System for Leasing Storage," bearing attorney docket no. 06502.0011-01000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,838, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," bearing attorney docket no. 06502.0011-02000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,834, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," bearing attorney docket no. 06502.0011-03000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,916, entitled "Leasing for Failure Detection," bearing attorney docket no. 06502.0011-04000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,933, entitled "Method for Transporting Behavior in Event Based System," bearing attorney docket no. 06502.0054-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,919, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," bearing attorney docket no. 06502.0062-01000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,938, entitled "Methods and Apparatus for Remote Method Invocation," bearing attorney docket no. 06502.0102-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/045,652, entitled "Method and System for Deterministic Hashes to Identify Remote Methods," bearing attorney docket no. 06502.0103-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,930, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," bearing attorney docket no. 06502.0105-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,917, entitled "Suspension and Continuation of Remote Methods," bearing attorney docket no. 06502.0106-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,835, entitled "Method and System for Multi-Entry and Multi-Template Matching in a Database," bearing attorney docket no. 06502.0107-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,839, entitled "Method and System for In-Place Modifications in a Database," bearing attorney docket no. 06502.0108, and filed on the same date herewith.

U.S. Patent Application No. 09/044,945, entitled "Method and System for Typesafe Attribute Matching in a Database," bearing attorney docket no. 06502.0109-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,931, entitled "Dynamic Lookup Service in a Distributed System," bearing attorney docket no. 06502.0110-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," bearing attorney docket no. 06502.0112-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," bearing attorney docket no. 06502.0113-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,932, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," bearing attorney docket no. 06502.0114-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on February 26, 1998.

U.S. Patent Application No. 09/044,936, entitled "An Interactive Design Tool for Persistent Shared Memory Spaces," bearing attorney docket no. 06502.0116-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,934, entitled "Polymorphic Token-Based Control," bearing attorney docket no. 06502.0117-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,915, entitled "Stack-Based Access Control," bearing attorney docket no. 06502.0118-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,944, entitled "Stack-Based Security Requirements," bearing attorney docket no. 06502.0119-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/044,837, entitled "Per-Method Designation of Security Requirements," bearing attorney docket no. 06502.0120-00000, and filed on the same date herewith.

BACKGROUND OF THE INVENTION

A. Field of the Invention

This invention generally relates to distributed systems and, more particularly, to a method and apparatus for determining a state of remote objects in a distributed system.

B. Description of the Related Art

Distributed systems typically comprise multiple machines, such as computers and related peripheral devices, connected in a network, for example, a Local Area Network (LAN), Wide Area Network (WAN), or the Internet. Distributed systems generally require that computational entities (e.g., applications, programs, applets,

etc.) running in different address spaces, potentially on different machines, be able to communicate.

For a basic communication mechanism, distributed object-oriented systems utilize Remote Method Invocation (RMI), which is more generally known as Remote Procedure Call (RPC). RMI facilitates application-level communication between "objects" residing in different address spaces.

In object-oriented systems, a "class" provides a "template" for the creation of "objects" (which represent items or instances manipulated by the system) having characteristics of that class. The term "template" denotes that the objects (i.e., data items) in each class, share certain characteristics or attributes determined by the class. Objects are typically created dynamically during system operation. Methods associated with a class are generally invoked (i.e., caused to operate) on the objects of the same class.

RMI is the action of invoking a method of a remote object. In response to the invocation of a method on a remote object using RMI, a lower level communications process causes the invoked method to be executed on the remote object.

The Java™ runtime system, which is designed to implement applications written in the Java™ object-oriented programming language, supports a specific Java™ RMI Application Program Interface (API). This API is explained in, for example, a document entitled "Remote Method Invocation Specification," Sun Microsystems, Inc. (1997), which is available via universal resource locator (URL) http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html, and is incorporated herein by reference. The Java language is described in many texts,

including one that is entitled "The Java Language Specification" by James Gosling, Bill Joy, and Guy Steele, Addison-Wesley, 1996. Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Java RMI assumes a homogeneous environment of the Java runtime system, and therefore Java RMI takes advantage of the object model for the Java language whenever possible. In the Java™ distributed object model, a remote object is one whose methods can be invoked from another Java runtime system, potentially on a different machine. A remote object is defined by one or more remote interfaces written in the Java language that specify the methods of the remote object. For example, interfaces enable entities invoking methods on remote objects to define the methods supported by the remote objects without specifying the implementation of those methods.

"Garbage collection" is the term used in technical literature and the relevant arts to refer to a class of algorithms utilized to carry out storage management, specifically automatic memory reclamation. There are many known garbage collection algorithms, including reference counting, mark-sweep, and generational garbage collection algorithms. These, and other garbage collection techniques, are described in detail in a book entitled "Garbage Collection, Algorithms For Automatic Dynamic Memory Management" by Richard Jones and Raphael Lins, John Wiley & Sons, 1996.

Distributed garbage collection extends the notion of garbage collection to the realm of distributed computing systems, reclaiming resources when no application on

any computer in a distributed system refers to them. An automated distributed garbage collection process frees the programmer from determining when it is safe to delete a remote object. In the absence of a distributed garbage collector (DGC), a remote object would need to keep track of all clients that refer to the object and the object storage can be reclaimed when all clients no longer reference that object. For this function, Java RMI's DGC relies on a reference-counting garbage collection algorithm similar to Modula-3's Network Objects. See "Network Objects" by Birrell, Nelson, and Owicki, Digital Equipment Corporation Systems Research Center Technical Report 115, 1994.

To accomplish reference-counting garbage collection, the Java RMI runtime system, which is an implementation of Java RMI on top of the Java runtime system, keeps track of all remote objects referenced by computational entities (i.e., clients) executing through a local virtual machine (VM). The Java™ VM (JVM) is an abstract computing machine of the runtime system that receives instructions from programs in the form of bytecodes and that interprets these bytecodes by dynamically converting them into a form for execution, such as object code, and executing them. The JVM is described in detail in a text entitled "The Java Virtual Machine Specification", by Tim Lindholm and Frank Yellin, Addison Wesley, 1996.

When a computational entity references a remote object, the local RMI runtime for the computational entity increments a corresponding reference count. Such a reference is typically referred to as a "strong" reference, and the computational entity is said to "hold" a strong reference to the remote object. A strong reference is one that will prevent the (remote) object from being collected. The first reference to a

remote object causes the runtime system to send a "referenced" message to the RMI runtime for that object (e.g., another machine in the distributed system holding the referenced object). As remote objects are found to be unreferenced in the local VM, the local RMI runtime decrements the corresponding reference count.

When the local VM discards the last reference to a remote object, an "unreferenced" message is sent to the RMI runtime corresponding to that object. When a remote object is not referenced by any client, the RMI runtime system for the remote object refers to its "local" object (which was considered a remote object for the client) using a weak reference. The weak reference allows the remote object's garbage collector to discard the object, provided no other local "strong" references to the object exist. As in the normal object life-cycle, a finalization process is called after the garbage collector determines that no more strong references to the object exist. One type of finalization process causes the memory allocated for an object to be returned to a memory heap for reuse. As long as a local "strong" reference to a remote object exists, it cannot be reclaimed in this way by a garbage collector and it can be passed in remote calls or returned to clients. Passing a remote object adds the identifier for the VM to which it was passed to the referenced set. As a result of the "referenced" call from the receiving VM, the RMI runtime will keep a "strong" reference to the remote object to prevent collection.

A remote object needing "unreferenced" notification, i.e., a notification that no more clients hold references, must implement a special interface referred to as the "java.rmi.server.Unreferenced" interface. In this manner, when all references to the object from remote entities (e.g., former clients of the object) no longer exist, a

method named "unreferenced" of the object will be invoked. The unreferenced method is called when the set of references for the object becomes empty.

Note that if a network partition exists between a client and a remote object, it is possible that premature collection of the remote object will occur since the transport might believe that the client crashed. Because of the possibility of premature collection, remote references cannot guarantee referential integrity; in other words, it is always possible that a remote reference may in fact not refer to an existing object. An attempt to use such a reference will generate a RemoteException error which must be handled by the computational entity making use of the reference.

Accordingly, there is a need for a system that enables computational entities to determine the state of remote objects. By obtaining such state information, computational entities can better manage references to remote objects and avoid unwanted RemoteExceptions without preventing the remote objects from being garbage collected.

SUMMARY OF THE INVENTION

Systems consistent with the present invention, as embodied and broadly described herein, a method is provided for selectively supplying a state change associated with remote objects in a distributed system. The method involves registering a request from a computational entity to receive notification as to a state change associated with a remote object. Registration of a notification request causes the creation of a remote weak reference to the remote object including an identifier of a location of the remote object.

Periodically, a request is sent to a location based on the identifier of the remote weak reference. When it is determined that a state change associated with the remote object has occurred, the registered computational entity is notified accordingly.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an implementation of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings,

FIG. 1. illustrates an exemplary network architecture in which systems consistent with the present invention may be implemented;

FIG. 2 is block diagram of an exemplary system architecture for a computer system with which the invention may be implemented;

FIG. 3 is a block diagram illustrating a data flow for handling remote objects in a distributed system consistent with the present invention;

FIGs. 4A and 4B are a flow chart illustrating acts performed by a client-side function of a distributed weak reference process consistent with an implementation of the present invention; and

FIG. 5 is a flow chart illustrating acts performed by a server-side function of a distributed weak reference process consistent with an implementation of the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to an implementation consistent with the present invention as illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings and the following description to refer to the same or like parts.

Introduction

Systems consistent with the present invention address shortcomings of the prior art and provide a method and apparatus for selectively supplying a state change associated with remote objects in a distributed system. The term "remote" is used herein to distinguish between an object located in an address space designated for a machine (such as a VM) operating in connection with a computational entity and an object located in an address space that is different from the address space designated for a machine operating in connection with the computational entity, the latter situation presenting a "remote" object. Consequently, an object located on the same physical machine as a computational entity can be considered a "remote" object with respect to that entity provided the computational entity uses an address space different from the space holding the object.

In general, a method and apparatus consistent with the present invention registers requests from computational entities to receive notification as to a "liveness" associated with remote objects. The term "liveness" is used herein to refer to whether a remote object is accessible at a specific time. A remote object is determined to be inaccessible when, for example, either a network partition prevents a computational entity from accessing the remote object or the remote object has been garbage

collected. Such a partition can occur as a result of a communication problem with the computational entity's machine that prevents it from accessing the machine associated with the remote object or a communication problem associated with the remote object's machine.

More specifically, when a computational entity references a remote object, the entity maintains a strong reference to the remote object. The computational entity may also register with the RMI runtime system to receive notification as to a change in the "liveness" of the remote object. Such a registration involves a registration-notification process of the RMI runtime system and causes the creation of a "remote weak reference," which is a reference to the remote object derived from the strong reference. The remote weak reference for the remote object will not prevent the remote object from being collected.

Based on a set of remote weak references, the RMI runtime system periodically sends status requests to machines associated with the corresponding remote objects. One or more of the responses to the requests are then provided to registered computational entities based on a change (if any) to the state of the remote object, i.e., the "liveness" of the remote object.

The Distributed System

Methods and systems consistent with the present invention operate in distributed systems comprised of, for example, multiple homogenous or heterogenous machines. An exemplary distributed system is shown in Fig. 1. This distributed system is generally comprised of various components, including both hardware and software. The exemplary distributed system (1) allows users of the system to share

services and resources over a network of many devices; (2) provides programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplifies the task of administering the distributed system. To accomplish these goals, the distributed system utilizes the Java™ programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, the distributed system is layered on top of the Java™ programming environment and exploits the characteristics of this environment, including the security offered by it and the strong typing provided by it. The programming environment is described more fully in Jaworski, Java 1.1 Developers Guide, Sams.net (1997), which is incorporated herein by reference.

In the exemplary distributed system, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, the distributed system provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. The distributed system may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Within the exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers,

displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system.

Both users and devices may join a Djinn. When joining a Djinn, the user or device adds zero or more services to the Djinn and may access any one of the services it contains. Thus, devices and users federate into a Djinn to share access to its services. The services of the Djinn appear programmatically as objects of the Java programming environment, which may include other objects, software components written in different programming languages, or hardware devices. A service has an interface defining the operations that can be requested of that service, and the type of the service determines the interfaces that make up that service.

Distributed system 100 shown in Fig. 1 is comprised of computer 102, computer 104, and device 106 interconnected by a network 108. Device 106 may be any of a number of devices, such as a printer, fax machine, storage device, or other devices. Network 108 may be a LAN, WAN, or the Internet. Although only two computers and one device are depicted as comprising distributed system 100, one skilled in the art will appreciate that distributed system 100 may include additional computers and devices or even computers alone without any devices.

Fig. 2 depicts computer 102 in greater detail to show a number of the software components of the distributed system 100. Computer 102 includes a memory 202, a secondary storage device 204, a central processing unit (CPU) 206, an input device 208, and a video display 210. Memory 202 includes a lookup service 212, a discovery server 214, an RMI 218, and a Java runtime system 216. Runtime system 216

includes a Java VM 220, and a garbage collector (GC) 224. Secondary storage device 204 includes a Java™ space 222.

As mentioned above, distributed system 100 is based on the Java programming environment and thus makes use of Java runtime system 216. Java runtime system 216 includes a Java™ API (not specifically shown), allowing programs running on top of Java runtime system 216 to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of an operating system (not shown) associated with computer 102. Since the Java API provides a single common API across all operating systems to which the Java programming environment is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. Java runtime system 216 is provided as part of the Java™ software development kit (JDK) available from Sun Microsystems of Mountain View, CA.

The Java virtual machine 220 also facilitates platform independence. The Java virtual machine 220 acts like an abstract computing machine, receiving instructions from programs in the form of bytecodes and interpreting these byte codes by dynamically converting them into a form for execution, such as object code, and executing them. Garbage collector (GC) 224 implements a garbage collection process to manage memory resources. GC 224 generally determines when an object is no longer referenced and initiates a process to reclaim the associated memory resources based upon the result of this determination.

RMI 218 facilitates remote method invocation by allowing objects executing on one computer or device to invoke methods of an object on another computer or device. Both RMI and the Java virtual machine are also provided as part of the Java software development kit. RMI 218 includes a distributed garbage collector (DGC) 226, which implements a reference-counting garbage collection algorithm similar to Modula-3's Network Objects.

Lookup service 212 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within distributed system 100. Lookup service 212 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. Lookup service 212 is described in greater detail in co-pending U.S. Patent Application No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," which has been previously incorporated herein by reference.

Discovery server 214 detects when a new device is added to distributed system 100, during a process known as boot and join or discovery, and when such a new device is detected, the discovery server passes a reference to lookup service 212 to the new device so that the new device may register its services with lookup service 212 and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in lookup service 212. The process of boot and join is described in greater detail in co-pending U.S. Patent Application No. 09/044,939, entitled "Apparatus and Method for

providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which has been previously incorporated herein by reference.

Java space 222 is an object repository used by programs within distributed system 100 to store objects. Programs use Java space 222 to store objects persistently as well as to make them accessible to other devices within the distributed system. Java spaces are described in greater detail in co-pending U.S. Patent Application No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, filed on November 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that distributed system 100 may contain more than one lookup service, discovery server, and Java spaces.

Framework for Distributed Weak References

Referring now to Fig. 3, a framework for using remote weak references in a distributed system to implement a registration-notification process for remote objects consistent with the present invention will be explained. Fig. 3 shows two computers 305 and 345, which may correspond to computers 102 and 104 shown in distributed system 100. Each computer, 305 and 345, includes a Distributed WEak Reference Process (DWERP) 320 and 360, respectively, to implement the registration-notification process for remote objects. DWERP 320 and 360 may be implemented in program code written in, for example, the Java programming language, and stored in memory 202 as part of RMI 218.

For purposes of this explanation, computer 305 will be referred to as a client because it comprises a computational entity, shown as process 310, that references a

remote object. In contrast, computer 345 will be referred to as a server because it comprises object 350, which constitutes a remote object with respect to process 310, which seeks status information on object 350 in this example. (The remote object may be located in Java space 222 of a distributed system conforming with the architecture shown in Figs. 1 and 2.) Although this description refers to the client and server as corresponding to different physical computers, it is also possible to configure a single computer operate as both a client and server. For example, a physical machine can be partitioned to support separate virtual machines and corresponding address spaces for the client and server.

Process 310 is said to hold a "strong" reference 330 to object 350 when process 310 receives a reference to object 350. A "strong" reference may be used to invoke methods on or pass as a parameter or return value in an RMI call. Although process 310 holds a "strong" reference to object 350 in this example, it is also possible that another computational entity in the distributed system holds a "strong" reference to object 350, yet process 310 seeks notification as to a status associated with object 350. In either case, process 310 registers a request with DWERP 320 to receive notification of a change in the "liveness" of object 350.

DWERP 320 monitors for the occurrence of events that affect the "liveness" of object 350, and notifies the registered process 310 accordingly. In one implementation, process 310 is only notified when an event affecting the "liveness" of remote object 350 is detected, although other implementations may permit process 310 to receive periodic updates on a status of the remote object.

If a network partition prevents process 310 from accessing object 350, DWERP 320 provides an event notification to process 310 because process 310 registered to receive such a notification. Such a partition can occur as a result of either a loss of connectivity between client 305 and the distributed system or a loss of connectivity between server 345 and the distributed system. In the event of a partition on the client side, DWERP 320 simply detects the event and notifies process 310 of the event. In the case of a network partition on the server side or a change in the "liveness" of object 350 resulting from its reclamation during a garbage collection cycle on server 345, DWERP 320 must communicate with DWERP 360 of server 345 to obtain information concerning these types of events affecting the "liveness" of object 350.

To that end, DWERP 320 periodically sends a "ping" or status request to DWERP 360 of computer 345. As shown in Fig. 3, DWERP 360 receives the ping from DWERP 320. In this example, remote weak reference 340 is used to determine the location of DWERP 360 and object 350 on server 345. Note that the process of pinging computers associated with remote objects is done asynchronously with respect to the registration process.

DWERP 360 of server 345 monitors resident or local objects, including object 350. Note that the DGC of the RMI for server 345 maintains the list of references to "local" objects designated as remote objects for RMI calls. DWERP 360 uses this information to monitor the local objects.

Based on the status of a selected object(s) identified by a ping, the server's DWERP 360 returns an appropriate response. In this example, DWERP 360 returns a

response to the ping from DWERP 320 indicating no change in the "liveness" associated with object 350. Since object 350 has not been garbage collected, and no network partition has occurred, the response from DWERP 360 in this example indicates that object 350 is still accessible. If, however, object 350 had been garbage collected by virtue of the fact that no more remote or local strong references to the object exist, the response from DWERP 360 would indicate such a status for object 350 in response to the ping from DWERP 320. In this case, DWERP 320 would notify process 310 of a change in the "liveness" of object 350 based on the response received from DWERP 360 of server 345. Thus, if the response indicates no change in the "liveness" of remote object 350, process 310 receives no event notification. In general, event notifications are provided only upon the occurrence and detection of an event changing the "liveness" of object 350.

As can be seen from the above discussion, the Distributed WEak Reference Process is comprised of two component functions. The first is a function performed on the client-side of a transaction and the second is a function performed on the server-side of a transaction. The client-side function of DWERP will be explained below with reference to the flow chart in Fig. 4 and the server-side function will be explained below with reference to the flow chart in Fig. 5.

Client-Side Function

As shown in Figs. 4A and 4B, the client-side function includes two aspects: registration and notification. The steps involved in registration are shown in Fig. 4A. First, the client-side DWERP receives a registration request from a computational entity that indicates that the entity wishes to receive a notification as to the status of a

selected remote object (step 410). The request includes either a strong reference to the selected remote object or a pointer to a location where the strong reference is located. Client-side DWERP then registers the request by creating a remote weak reference to the selected remote object (step 412).

The notification aspect of the client-side DWERP is shown in Fig. 4B. Periodically, and not necessarily in sync with the registration aspect, the client-side DWERP accesses a set of remote weak references for remote objects to identify the location of any remote objects with corresponding notification requests from computational entities, i.e., the server for the remote object (step 420), and sends a ping to each identified server with information on the remote object for which a liveness update has been requested (step 430).

The client-side DWERP then determines whether there has been a change in the "liveness" of each remote object with a registered remote weak reference (step 445) and notifies the corresponding registered computational entity when such a change has occurred (step 450). If there has been no change in the liveness of a remote object (step 445), client-side DWERP returns to a wait state (step 460) where it remains until it is time for another ping cycle. In one cycle the client-side DWERP may simultaneously send pings to servers for all remote objects with registered remote weak references or it may use each cycle to send a ping to only one or a limited number of servers.

If, however, an event affecting the "liveness" of a remote object is detected, then the corresponding registered computational entity is provided a notification of the event (step 450). For example, if a network partition preventing the client from

sending pings or receiving responses to pings has occurred, the client-side DWERP provides an event notification to one or more computational entities with registered requests to receive an event notification when an event affecting the "liveness" of a remote object has occurred. Similarly, the DWERP notifies computational entities when the DWERP does not receive a response to a ping (which indicates a communication problem with the server for a remote object), or when the server's DWERP responds with an indication that the remote object has been garbage collected.

In this manner, the client-side function handles the registration by computational entities for status on remote objects and the notification of those registered computational entities upon receipt of information on the status of selected remote objects.

Server-Side Function

As shown in Fig. 5, the server-side function of the DWERP involves receiving each ping from a client machine (step 510), determining whether there has been a change in the "liveness" of an identified local object for the server, e.g., has the object been garbage collected (step 520), and sending a response, including any change in the "liveness" that object, to the client machine (step 530).

Although this description describes the client and server functions of a DWERP separately, those skilled in the art will recognize that both functions of a DWERP would likely be present on each machine in a distributed system to take full advantage of the concepts of the present invention.

Conclusion

Systems consistent with the present invention thus implement a methodology for determining a change in the liveness of remote objects in a distributed system. In summary, computational entities register requests to receive notification as to a change in the liveness of selected remote objects. Status requests on the remote object are periodically sent to the server where the remote object resides. The remote object is located using a remote weak reference to the remote object. Responses to the status requests are then provided to the registered computational entity based on the occurrence of an event that changes the liveness of the remote object. In this manner, computational entities can receive a notification as to a state change of remote objects.

The foregoing description of an implementation of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. The invention may be implemented with both object-oriented and non-object-oriented programming systems.

Although systems and methods consistent with the present invention are described as operating in the exemplary distributed system and the Java programming environment, one skilled in the art will appreciate that the present invention can be practiced in other systems and programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled

in the art will appreciate that these aspects can also be stored on other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM. The scope of the invention is defined by the claims and their equivalents.

WHAT IS CLAIMED IS:

1. A method for supplying a state change associated with remote objects in a distributed system comprised of multiple platforms, the method comprising:

providing an object resident on one of the platforms;

registering a request from a computational entity located on one of the platforms remote with respect to the one of the platforms upon which the object resides to receive an indication as to a state change associated with the object;

determining whether an event has occurred changing the state associated with the object; and

providing a notification to the computational entity based on the determination.

2. The method of claim 1, wherein the step of providing a notification to the computational entity based on the determination includes

supplying the notification only when it is determined that the event has occurred.

3. The method of claim 1, wherein the step of determining whether an event has occurred changing the state associated with the object includes

sending a status request to the one of the platforms upon which the object resides.

4. The method of claim 1, wherein the step of determining whether an event has occurred changing the state associated with the object includes

ascertaining whether a network partition separates the platform associated with the computational entity and the platform associated with the object.

5. The method of claim 1, wherein objects have corresponding references indicating the location of objects within the distributed system, and wherein the step of determining whether an event has occurred changing the state associated with the object includes

locating the one of the platforms upon which the object resides based on a reference corresponding to the object; and

sending a status request to the located platform.

6. The method of claim 1, wherein the step of determining whether an event has occurred changing the state associated with the object includes

receiving a status indicator from the one of the platforms upon which the object resides.

7. The method of claim 1, wherein the step of providing a notification to the computational entity based on the determination includes

returning a notification as to a liveness of the object to the computational entity.

8. The method of claim 1, wherein computational entities maintain references indicating the location of objects within the distributed system, and wherein the registering step includes

receiving a reference to the object from the computational entity, and

creating a remote weak reference corresponding to the object based on the received reference; and

wherein the step of determining whether an event has occurred changing the state associated with the object includes

locating the one of the platforms upon which the object resides based on the remote weak reference corresponding to the object; and

sending a status request to the located platform.

9. A method for determining a state change associated with objects in a distributed object-oriented system comprised of multiple address spaces, wherein computational entities operate in connection with the address spaces, the method performed by a processor comprising:

providing a computational entity operating in connection with one of the address spaces;

providing a reference to an object associated with an address space that is remote with respect to the address space associated with the computational entity;

registering a request from the computational entity for notification of a state change associated with the object; and

determining whether an event has occurred altering a state associated with the object.

10. The method of claim 9 wherein the determining step includes

receiving an indication that the state associated with the object has been altered.

11. The method of claim 9, further comprising

supplying a notification to the computational entity only when it is determined the state associated with the object has been altered.

12. The method of claim 9, wherein the step of determining whether an event has occurred altering a state associated with the object includes

accessing the address space associated with the object.

13. The method of claim 9, wherein the step of determining whether an event has occurred altering a state associated with the object includes

ascertaining whether a network partition separates the address space associated with the computational entity and the address space associated with the object.

14. The method of claim 9, wherein objects have corresponding references indicating their address space within the distributed system, and wherein the step of

determining whether an event has occurred altering a state associated with the object includes

locating the address space associated with the object based on a reference corresponding to the object; and

accessing the address space associated with the object.

15. The method of claim 9, further comprising

returning a liveness notification to the computational entity when it is determined the state associated with the object has been altered.

16. A method for supplying a state change associated with objects in a distributed system comprised of multiple platforms, the method performed by one of the platforms comprising:

receiving a status request from a remote platform having a reference to an object;

determining whether an event has occurred altering a state associated with the object; and

returning a notification to the remote platform based on the determination.

17. A method for supplying a state change associated with objects in a distributed system, the method comprising:

providing a remote object;

creating a reference to the remote object including an identifier of a location of the remote object;

registering a request to receive notification when a remote object has been collected;

periodically sending a status request based on the identifier of the location of the remote object;

determining whether the remote object is queued for collection or has been collected in response to receipt of one of the status requests; and

providing a notification when it is determined that the remote object is queued for collection or has been collected in response to the registered request.

18. A method for determining a state change of remote objects in a distributed system comprised of multiple address spaces, the method comprising:

providing a remote object resident on a first address space;

creating a reference to the remote object on a second address space including an identifier for the first address space;

registering a request from a process resident on the second address space to receive notification when the remote object has been garbage collected;

periodically sending a status request to the first address space to determine the status of the remote object;

determining whether the remote object has been garbage collected in response to receipt by the first address space of one of the status requests; and

providing a notification that the remote object has been garbage collected to the process in response to the registered request.

19. A system for supplying a state change associated with remote objects in a distributed environment comprised of multiple platforms, the system comprising:

a first platform, comprising

a memory having an object; and

a second platform remote with respect to the first platform, comprising

a memory having program instructions, and

a processor configured touse the program instructions to

register a request from a computational entity to receive an indication as to a state change associated with the object,

determine whether an event has occurred changing the state associated with the object, and

provide a notification to the computational entity based on the determination.

20. The system of claim 19, wherein the processor is further configured to to use the program instructions to supply the notification only when it is determined that the event has occurred.

21. The system of claim 19 wherein the processor is further configured to touse the program instructions to ascertain whether a network partition separates the platform associated with the computational entity and the memory.

22. An apparatus for determining a state change associated with objects in a distributed object-oriented system comprised of multiple address spaces, wherein a computational entity operates in connection with one of the address spaces, and wherein a reference is provided to an object associated with an address space that is remote with respect to the address space associated with the computational entity, the apparatus comprising:

a registration component configured to register a request from the computational entity for notification of a state change associated with the object; and

a processor configured to determine whether an event has occurred altering a state associated with the object.

23. The apparatus of claim 22 wherein the processor includes

a receiver configured to receive an indication that the state associated with the object has been altered.

24. The apparatus of claim 22. further comprising

a notification component configured to supply a notification to the computational entity only when it is determined the state associated with the object has been altered.

25. The apparatus of claim 22, wherein the processor includes

a component configured to access the address space associated with the object.

26. The apparatus of claim 22, wherein the processor includes

a component configured to ascertain whether a network partition separates the address space associated with the computational entity and the address space associated with the object.

27. The apparatus of claim 22, wherein objects have corresponding references indicating their address space within the distributed system, and wherein the processor includes

a component configured to locate the address space associated with the object based on a reference corresponding to the object; and

a component configured to access the address space associated with the object.

28. The apparatus of claim 22, further comprising

a notification component configured to return a liveness notification to the computational entity when it is determined the state associated with the object has been altered.

29. A system for supplying a state change associated with objects in a distributed system comprised of multiple platforms, the system comprising:

a memory having program instructions; and

a processor configured to use the program instructions to

receive a status request from a remote platform having a reference to an object;

determine whether an event has occurred altering a state associated with the object; and

return a notification based on the determination.

30. A computer-readable medium containing instructions for supplying a state change associated with remote objects in a distributed system comprised of multiple platforms, by:

providing an object resident on one of the platforms;

registering a request from a computational entity located on one of the platforms remote with respect to the one of the platforms upon which the object resides to receive an indication as to a state change associated with the object;

determining whether an event has occurred changing the state associated with the object; and

providing a notification to the computational entity based on the determination.

31. The computer-readable medium of claim 30, wherein the step of providing a notification to the computational entity based on the determination includes

supplying the notification only when it is determined that the event has occurred.

32. The computer-readable medium of claim 30, wherein the step of determining whether an event has occurred changing the state associated with the object includes sending a status request to the one of the platforms upon which the object resides.

33. The computer-readable medium of claim 30, wherein the step of determining whether an event has occurred changing the state associated with the object includes

ascertaining whether a network partition separates the platform associated with the computational entity and the platform associated with the object.

34. The computer-readable medium of claim 30, wherein objects have corresponding references indicating the location of objects within the distributed system, and wherein the step of determining whether an event has occurred changing the state associated with the object includes

locating the one of the platforms upon which the object resides based on a reference corresponding to the object; and

sending a status request to the located platform.

35. The computer-readable medium of claim 30, wherein the step of determining whether an event has occurred changing the state associated with the object includes

receiving a status indicator from the one of the platforms upon which the object resides.

36. The computer-readable medium of claim 30, wherein the step of providing a notification to the computational entity based on the determination includes

returning a notification as to a liveness of the object to the computational entity.

37. The computer-readable medium of claim 30, wherein computational entities maintain references indicating the location of objects within the distributed system, and wherein the registering step includes

receiving a reference to the object from the computational entity, and

creating a remote weak reference corresponding to the object based on the received reference; and

wherein the step of determining whether an event has occurred changing the state associated with the object includes

locating the one of the platforms upon which the object resides based on the remote weak reference corresponding to the object; and

sending a status request to the located platform.

38. A computer-readable medium containing instructions for determining a state change associated with objects in a distributed object-oriented system comprised of multiple address spaces, wherein computational entities operate in connection with the address spaces, by:

providing a computational entity operating in connection with one of the address spaces;

providing a reference to an object associated with an address space that is remote with respect to the address space associated with the computational entity;

registering a request from the computational entity for notification of a state change associated with the object; and

determining whether an event has occurred altering a state associated with the object.

39. The computer-readable medium of claim 38 wherein the determining step includes receiving an indication that the state associated with the object has been altered.

40. The computer-readable medium of claim 38, further comprising

supplying a notification to the computational entity only when it is determined the state associated with the object has been altered.

41. The computer-readable medium of claim 38, wherein the step of determining whether an event has occurred altering a state associated with the object includes

accessing the address space associated with the object.

42. The computer-readable medium of claim 38, wherein the step of determining whether an event has occurred altering a state associated with the object includes ascertaining whether a network partition separates the address space associated with the computational entity and the address space associated with the object.

43. The computer-readable medium of claim 38, wherein objects have corresponding references indicating their address space within the distributed system, and wherein the step of determining whether an event has occurred altering a state associated with the object includes

locating the address space associated with the object based on a reference corresponding to the object; and

accessing the address space associated with the object.

44. The computer-readable medium of claim 38, further comprising returning a liveness notification to the computational entity when it is determined the state associated with the object has been altered.

45. An apparatus for supplying a state change associated with objects in a distributed system comprised of multiple platforms, comprising:

means for receiving a status request from a remote platform having a reference to an object;

means for determining whether an event has occurred altering a state associated with the object; and

means for returning a notification based on the determination.

46. A memory for storing data for access by a computational entity being executed by a processor, wherein

an object residing in a location remote with respect to the computational entity, and

the object becomes reclaimable when a set of references to the object is empty, the memory comprising:

a remote weak reference to the object distinct from any reference in the set of references to the object, wherein

the remote weak reference enables the computational entity to obtain information corresponding to an event associated with the object without

preventing the object from being reclaimed when the set of references to the object is empty.

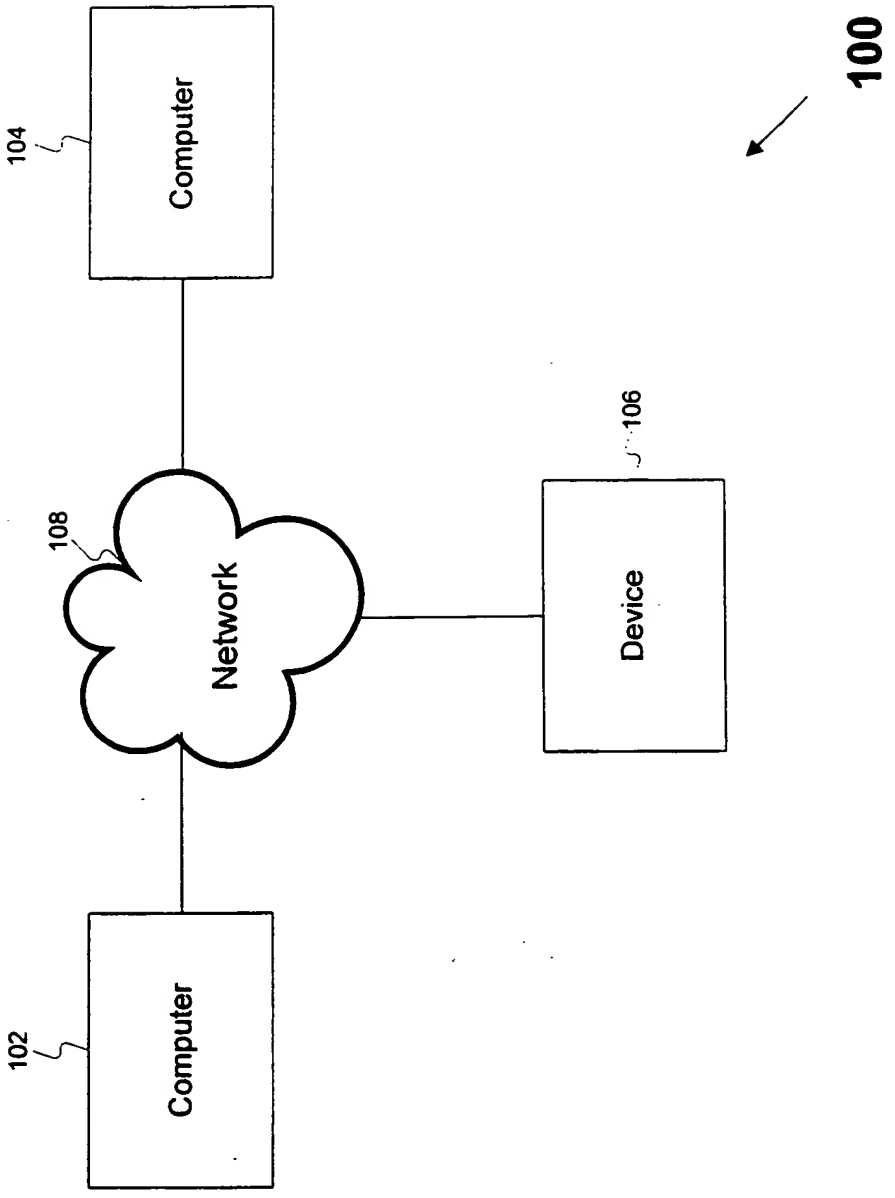


FIG. 1

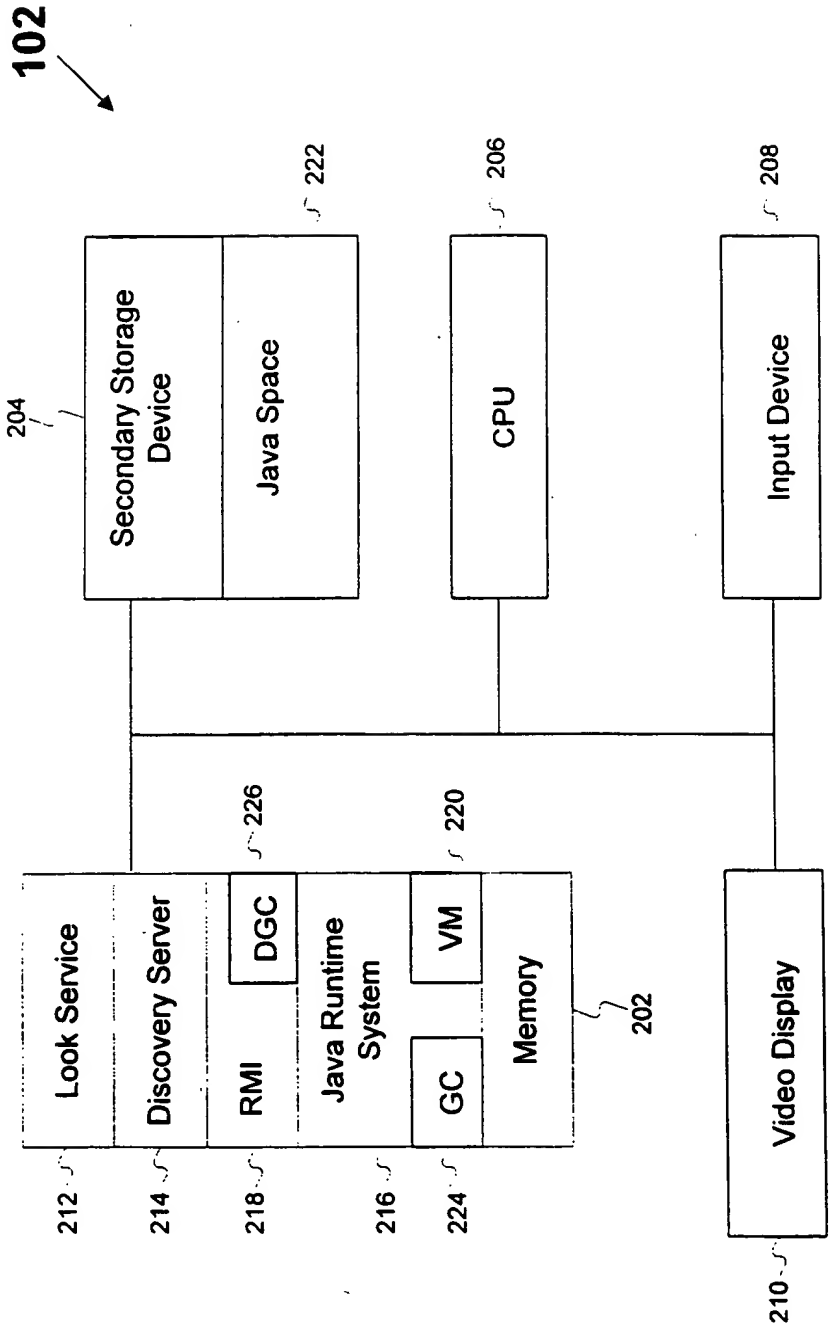

102
204
Secondary Storage Device
222
Java Space
206
CPU
208
Input Device
226
220
Look Service
Discovery Server
RMI
DGC
Java Runtime System
VM
GC
Memory
202
Video Display
212
214
218
216
224
210

FIG. 2

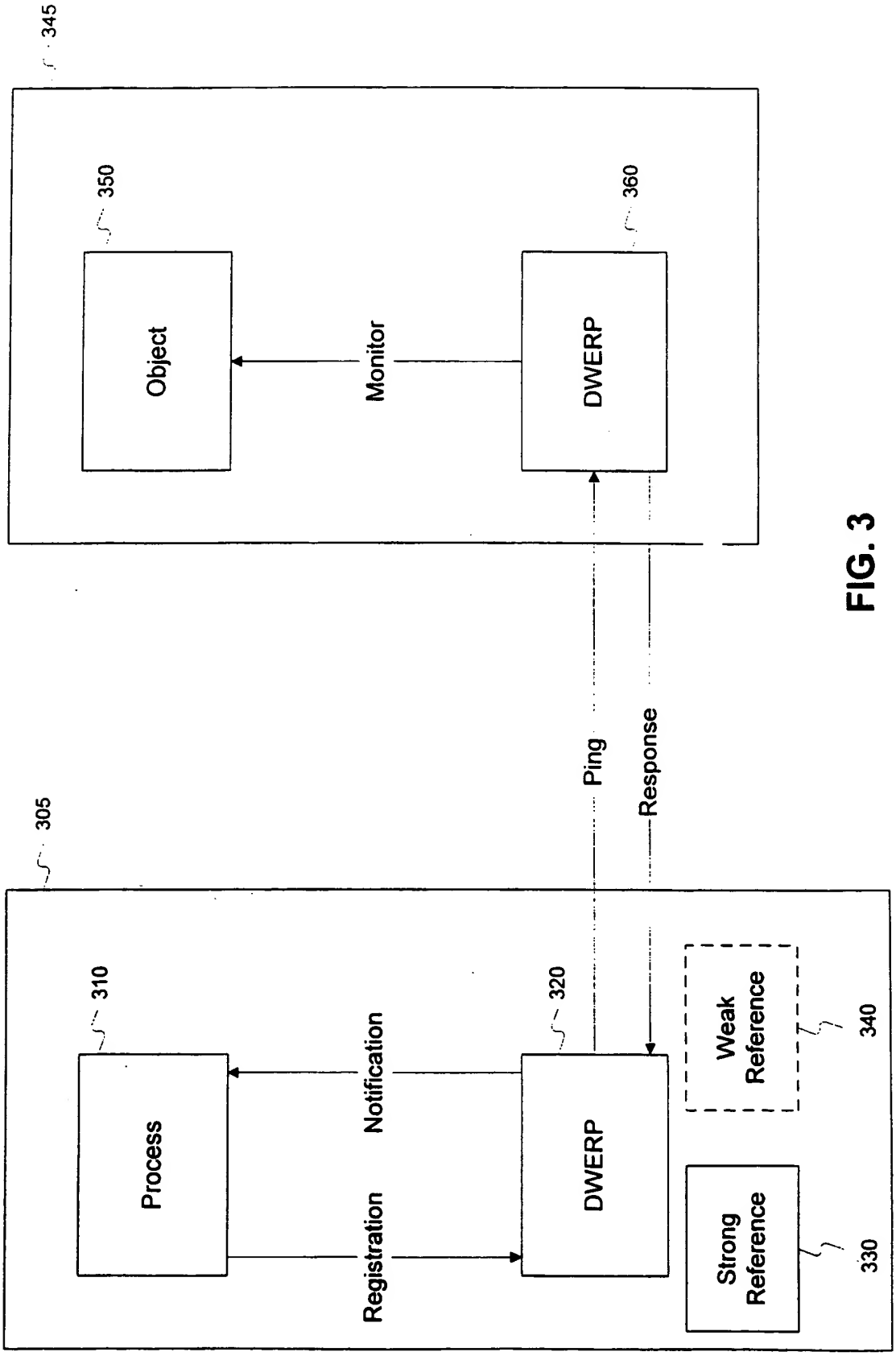

345
350
360
Object
Monitor
DWERP
Ping
Response
305
310
320
340
Process
Notification
Registration
DWERP
Weak Reference
Strong Reference
330

FIG. 3

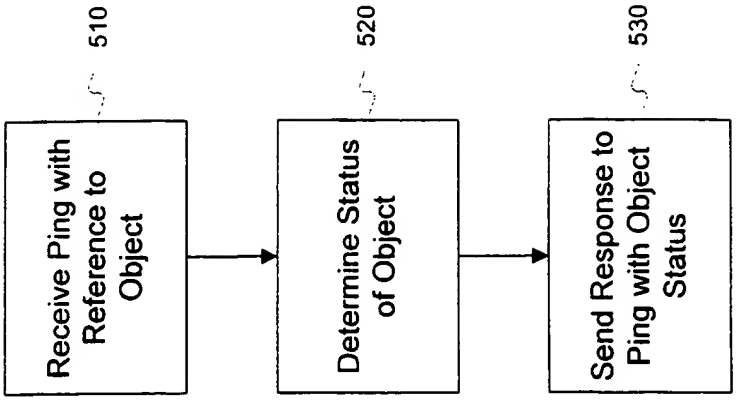

510
Receive Ping with Reference to Object
520
Determine Status of Object
530
Send Response to Ping with Object Status

FIG. 5

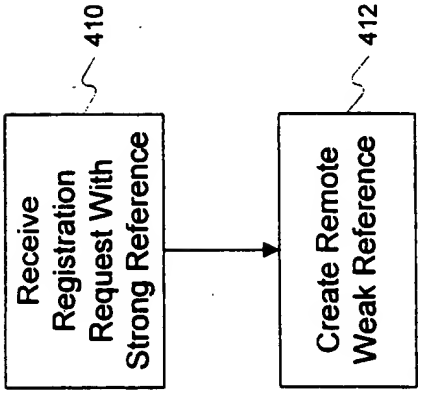
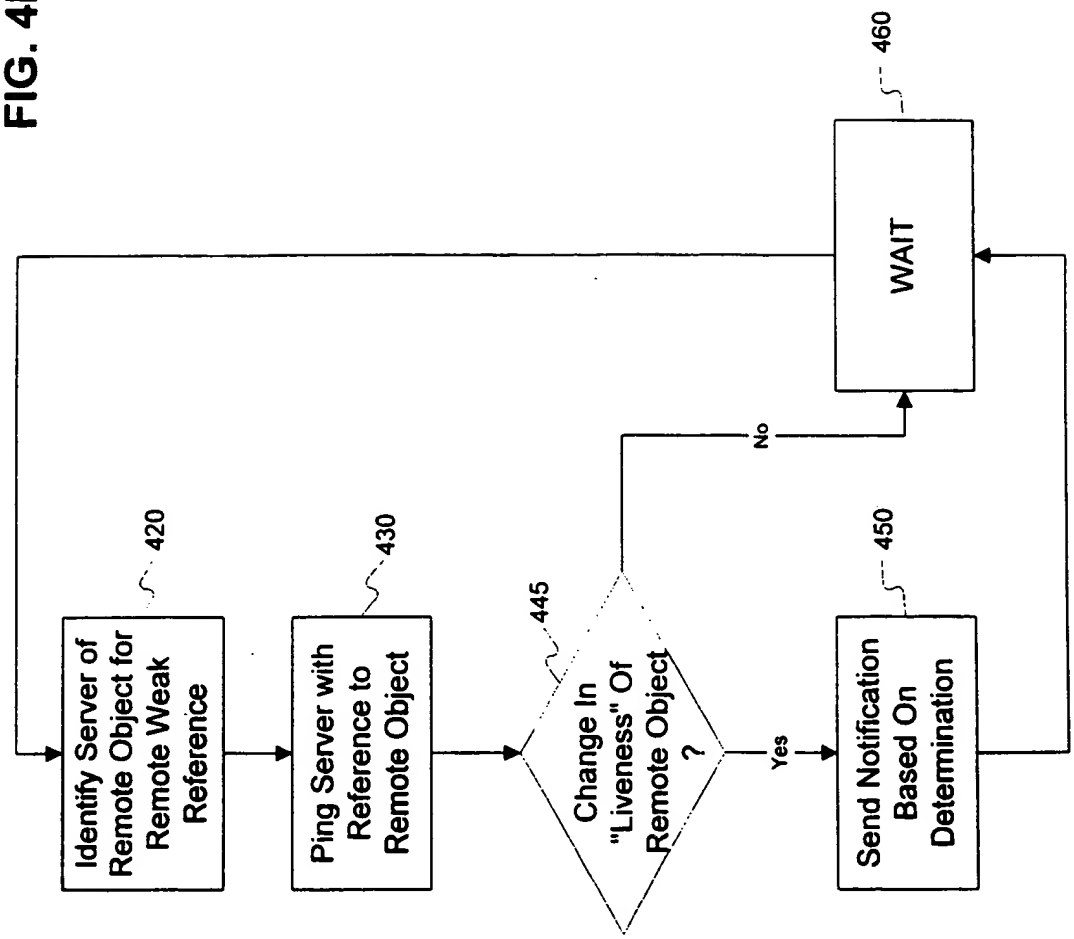

410
Receive Registration Request With Strong Reference
412
Create Remote Weak Reference

FIG. 4A

FIG. 4B


Identify Server of Remote Object for Remote Weak Reference
420
Ping Server with Reference to Remote Object
430
Change In "Liveness" Of Remote Object ?
445
No
Yes
Send Notification Based On Determination
450
WAIT
460

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 99/03944

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 109 486 A (SEYMOUR LESLIE G) 28 April 1992 (1992-04-28)	1,2,6,7, 9-12,14, 15,19, 20, 22-25, 27,28, 30,31, 35,36, 38-41, 43,44
Y	column 9, line 34 - column 12, line 19	3,5,32, 34
A		8,17,18, 37
	--- -/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 July 1999	09/08/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bijn, K

2

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 99/03944

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 390 328 A (FREY JEFFREY A ET AL) 14 February 1995 (1995-02-14)	1,2,6,7, 9-11,15, 19,20, 22-24, 28,30, 31,35, 36, 38-40,44
A	column 4, line 43 - column 6, line 11 column 8, line 8 - column 10, line 34 column 13, line 52 - column 14, line 13	3,5,8, 16-18, 29,32, 34,37,45
X	EP 0 697 655 A (CANON KK) 21 February 1996 (1996-02-21)	16,29,45
A	column 6, line 6 - column 8, line 45	1,6-10, 15, 17-19, 22,23, 28,30, 35-39,44
Y	column 11, line 18 - column 12, line 14	3,5,32, 34
X	WALDO J ET AL: "Events in an RPC based distributed system" PROCEEDINGS OF THE 1995 USENIX TECHNICAL CONFERENCE, PROCEEDINGS USENIX WINTER 1995 TECHNICAL CONFERENCE, NEW ORLEANS, LA, USA, 16-20 JAN. 1995, pages 131-142, XP002109939 1995, Berkeley, CA, USA, USENIX Assoc, USA	1,9,19, 22,30,38
A	page 133, right-hand column, line 32 - page 135, left-hand column, last last	2,10,11, 17,18, 20,23, 24,31, 39,40
X	EP 0 767 432 A (SUN MICROSYSTEMS INC) 9 April 1997 (1997-04-09)	46
A	column 7, line 51 - column 9, line 27	1,8,9, 17-19, 22,30, 37,38

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 99/03944

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5109486 A	28-04-1992	NONE	
US 5390328 A	14-02-1995	NONE	
EP 0697655 A	21-02-1996	JP 8123715 A	17-05-1996
EP 0767432 A	09-04-1997	US 5765174 A	09-06-1998
		AU 6425296 A	10-04-1997
		JP 9185552 A	15-07-1997